A LESSON IN BASIC PROGRAMMING

24706/8311

Printed in South Africa

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 6

A LESSON IN BASIC PROGRAMMING

Edition 2

STUDY UNIT 6

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

☐ Code a BASIC program from a good, working design and produce accurate software . . . . **Page 1**

☐ Understand how programs can quickly be written to display meaningful output with only a few, simple instructions and how variable names are used to store data entered from the keyboard and then printed onto the CRT . . . . **Pages 2–6**

☐ Use the GOTO BASIC keyword which allows for both the repeating and the skipping of program instructions . **Pages 6–9**

☐ Terminate a program loop by using the keyword IF . . . . **Pages 9–11**

☐ Insert remarks into programs which will make the software easier to understand and to modify . . . . **Page 11**

☐ Use the LET statement in BASIC to give a value to a variable . . . . **Page 13**

☐ Use the arithmetic operations (adding, subtracting, multiplying, dividing and raising to a power) to allow the computer to solve formulas . . . . **Pages 15–16**

☐ Solve equations which mix several arithmetic operations together by using parentheses to arrive at the correct answer . . . . **Pages 16–17**

☐ Assign variables to serve as accumulators so that totals may be taken . . . . **Pages 17–19**

☐ Design your own BASIC programs by closely following the logic explained and illustrated . . . . **Pages 19–23**

LEARNING AIDS
Programmer's Check #1 . . . . . **12**
Programmer's Check #2 . . **24–25**

EXAM 6 (Examination for Study Unit 6) **31–35**
ANSWER SHEET **37**

STUDY UNIT 6

A LESSON IN BASIC PROGRAMMING

DO YOU KNOW?

- **How a program can be documented?**
- **How punctuation can control output?**
- **The difference between coding a loop and being in a loop?**
- **How the computer evaluates an arithmetic formula?**

FIGURE 1—The data produced by a very simple, computerized accounting program can lead to extraordinary debates in the board room of the corporation. Questions asked here often lead to new programming assignments in a quest for more precise information.

CODING FROM DESIGN

"To err is human: to really foul things up requires a computer"—Anonymous

A computer is a powerful, accurate and lightning-fast technological wonder. It is capable of doing complex calculations in less than a second. But it can only do what we tell it! The "brains" of the computer are essentially non-intelligent—merely robot-like.

You will find that it takes longer to *design* a good program than it does to actually *write* one. Throughout this Study Unit, it is critically important that we first know what we want the computer to do before we command the computer to do it!

The basic development cycle (analysis, design, coding, testing), when strictly followed, will produce accurate, efficient and easily maintainable programs. Throughout this Study Unit, we will be translating our design (a flowchart) into the BASIC language.

This is not the *only* way in which a working program can be produced; it is, however, the *best* way! Writing a program without an accurate flowchart is not far removed from writing a program by randomly striking keys on the keyboard; given an infinite amount of time, it will eventually work. But, by applying sound logic, you may be surprised at how little time it takes to produce professional and satisfying software.

CONSTANT OR VARIABLE???

CONSTANT = A value such as a name which does not change during execution of the program. In PRINT statements, constants are placed inside quotation marks.

Example: 1Ø PRINT "NAME"

VARIABLE = A value such as a number, name, or character which changes during execution of the program.

Example: 1Ø REM A$...AREA CODE
2Ø LET R = .Ø8

FIGURE 2—Understanding how certain values are constant and others are variable is essential when programming.

INPUT/OUTPUT PROGRAMMING

In most applications, data submitted as input to the computer is reformated by a program in order to produce a display of the output. In this section, we will be introduced to those BASIC statements necessary to produce an input/output program.

Input

The computer is only able to print data that has been submitted to it. That data may be available to the program in three different ways:

1. The data may be located within the program itself as DATA statements. If so, the keyword READ will access this data.
2. The data may be stored on an external file on punched cards, magnetic tape or magnetic disk. In this case, the READ statement must reference this file.
3. The data can be submitted as input while the program is running; this is what is meant by interactive programming. An INPUT statement will accomplish this.

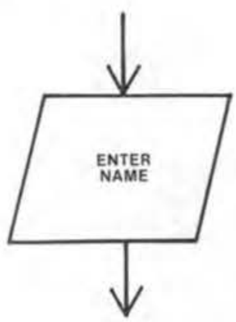
The first two methods are not available on all microcomputers. (BASIC is not a rigidly standardized language. Consult your technical manual to see if you can use the READ and DATA statements.)

In this Study Unit we will use the INPUT keyword to enter our data. The format of the INPUT is as follows:

nn INPUT variable-name

where: nn stands for a line number
INPUT is the BASIC keyword
and, variable-name represents the field we wish to have submitted as input.

An INPUT statement is coded within a BASIC program whenever our flowchart design looks like this:



The INPUT statement, when encountered in a program, will accept the characters entered by the operator (in this case, you!) and store that data in consecutive bytes of RAM at a specific address. The characters are "held" in their encoded (EBCDIC or ASCII) values.

Variables

A variable name is the way that data is assigned by the computer to an address in RAM. Rather than referencing BYTE -1Ø24, for example, we can call the data stored at this location "A" or "X." There are two different categories of variable names: string and numeric.

String variables are fields which can contain any values during execution— that is, letters of the alphabet, numbers or special characters (e.g., $, ?, #, ', `, etc.). Variables which will contain values representing names, addresses, etc., must be defined as string variables.

Numeric variables are fields which will contain numbers *only*. Arithmetic operations require numeric data. Therefore, no string variables can be used in a calculation.

SYNTAX

TAB 3; N$; TAB 2Ø; S$; TAB Ø; N$; TAB 13; A$; TAB 22; P$; TAB...; = TAB

FIGURE 3—Unlike normal writing, computer programmers use punctuation marks in special ways. The semicolon, for example, is used by the computer to separate tabular columns of variables. Commas, on the other hand, are used very carefully to separate items in a print field.

In BASIC, variables are assigned "attributes" (as either number or string variables) according to the names given to them. Different versions of BASIC have varying rules for assigning variable names. (Consult your technical manual. The following rules are valid on many computers.)

Variable names begin with a letter of the alphabet (A through Z). The second character of a variable name determines its attributes. If this character is a dollar sign ("$"), then the variable name is a string variable (which can contain any values but can't be used in arithmetic).

If the second character is a number, or is not present at all, then the name stands for a numeric variable (which can contain only numbers, and can be used in arithmetic).

Let's look at some examples:

1. Valid string variable names:
   A$, N$, U$, etc.

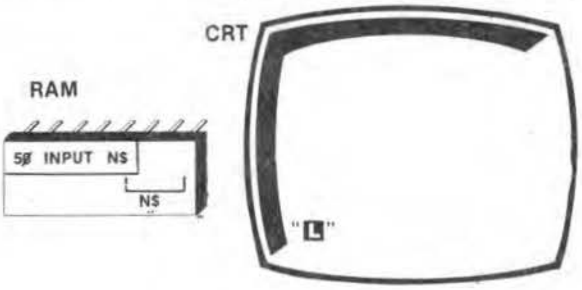
2. Valid numeric variable names:
   A1, Z, F2, etc.

Programmers determine the variable names they wish to use in their programs. The computer does not "understand" what these names mean; it merely sets aside a group of bytes in RAM to store values in. However, be advised that it is best whenever possible to choose variables which "mean" something to you, such as letting "A$" be the field name for an Address field; C representing a Check amount, etc.

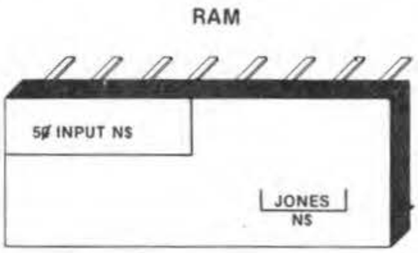
In the program we will soon be developing, it will be necessary for us to input three variables into main storage: a name field, an address field, and a city/state field. One of our BASIC statements which will allow us to do this is:

```
5Ø INPUT N$
```

When this statement is encountered during execution of the program, we, the user, will be "prompted" to give the computer a name.



If we were to press the keys "JONES" and press ENTER, these characters will be encoded and stored in consecutive bytes of RAM.



Remember, since N$ is a string variable, we could input *any* characters into main storage.

Whenever character data is used, it must be enclosed within quotation marks. That is why the "L" cursor is automatically displayed within quotes on most computers. If we were to input data into a numeric variable, no quote marks would appear and we would be restricted to entering numbers only.

Output

The other half of an input/output program is printing or displaying the data entered. The PRINT statement will be used for this. The format of the PRINT statement is:

nn PRINT variable-name(s) or constant(s)

where: nn is the line number, PRINT is the keyword

variable-name(s) or constant(s) stands for the values we wish to display.

We may always display the values contained in the variables we have given as input. For example, the BASIC statement

```
6Ø PRINT N$
```

will display upon the CRT the name entered by our previous 5Ø INPUT N$ statement.

Enter the following BASIC program and see how these statements can be used together:

```
1Ø INPUT N$
2Ø PRINT N$
3Ø STOP
```

When you have entered this program, RUN it.

When the INPUT statement is encountered, the program waits for you to enter some characters. Put in your name and then press ENTER.

You should see your name on the screen and a report code indicating that the program STOPped at line 3Ø.

Try RUNning it again, this time using another name. See how it works?

Let's add another variable to this program, a social security number. Since our social security numbers have hyphens in them (e.g., ØØ5-99-99ØØ), and hyphens are not numbers, we will call this variable name "S$." We will INPUT this variable and PRINT it as well.

Modify the first program (or reenter it) to look like this:

```
1Ø INPUT N$
15 INPUT S$
2Ø PRINT N$
25 PRINT S$
3Ø STOP
```

RUN it. This time you are prompted twice. The first time, ENTER your name; the second time, ENTER your social security number. Your screen should now display both values; first, the name entered and, your social security number on the next line of the screen. RUN it again. Enter some "nonsense" characters into these two variables. See, the computer is not as "smart" as some people think. (Remember GIGO, garbage in-garbage out.)

Punctuation in the PRINT statement affects the way our screen will display the output. On most computers, the screen is divided into print zones of about sixteen positions each. A computer capable of displaying 8Ø characters on 16 lines might be divided into five zones (8Ø/16 = 5); a 32-character line into two zones (32/16 = 2). (Check your manual to see how many print zones your computer uses. For our purpose here, two print zones per line are assumed.)

Once again, let's modify our program to demonstrate the use of print zones.

ENTER this program:

```
1Ø INPUT N$

15 INPUT S$

2Ø PRINT N$, S$

3Ø STOP
```

RUN the program and submit your name and social security number. Note that, this time, both values are printed on the same line of the CRT, but each in its own print zone.

Semicolons (;) can also be used in a PRINT statement. But unlike commas, semicolons cause the second value to be displayed immediately to the right of the first value.

EDIT the program just mentioned, replacing the comma in statement 2Ø with a semicolon:

```
2Ø PRINT N$; S$
```

RUN it. Now we have our name immediately followed by our social security number.

Another way of formating output is by using the TAB function in our PRINT statement. The format of a PRINT statement which uses the TAB function is:

nn PRINT TAB C1; variable-1
[;TAB C2; variable-2]

where: **nn is the line number**
PRINT is the keyword
TAB is the function
C1 and C2 are the column numbers where we wish to have variable-1 and variable-2 displayed.

Enter the following program:

```
1Ø INPUT N$
15 INPUT S$
2Ø PRINT TAB 3; N$; TAB 2Ø; S$
3Ø STOP
```

In order to enter the TAB function on line 2Ø, your keyboard has to be shifted into the FUNCTION mode.

Key in line number 2Ø and the PRINT keyword. Then, holding down the SHIFT key, press the ENTER key. Your "L" cursor should now be an "F" cursor. The next key you press will be assumed to have the value underneath that key. By pressing key #20, the word TAB should appear. Enter the number 3, a semicolon, the variable name "N$," and another

semicolon. Then shift into the "function" mode again, put in another TAB, the number 2Ø, a semicolon and lastly the variable name "S$."

NOTE: When using a lot of punctuation in a statement, it's easy to make syntax errors. Examine your statement carefully to ensure that you have the semicolons in the right places.

RUN this program. This time, your name should appear in column number 3 and your social security number in column 2Ø. (On most computers, columns are numbered starting with Ø so that these variables are actually in the fourth and twenty-first columns of the line.)

Experiment with other TAB numbers.

The TAB function is a very convenient way of arranging output on the CRT, and we will use this function often in the sample programs that follow.

Constants

A constant is a value which does not change during execution of the program. We can display constant data on our screen by enclosing the values within quotes on a PRINT statement. For example, if we wished to display headings above the values we have been printing, we could do so in the following manner:

```
1Ø INPUT N$
15 INPUT S$
18 PRINT "NAME"; "SOC.SEC.NO."
2Ø PRINT N$; S$
3Ø STOP
```

When we RUN this program, line 18 will print the same characters as we have enclosed within quotes (constants) on one line, and the values we entered for our variables on the next line.

A mixture of constants and variables can be printed on the same line.

Try this program:

```
1Ø INPUT N$
15 INPUT S$
18 PRINT "NAME", N$
2Ø PRINT "SOC.SEC.NO.", S$
3Ø STOP
```

Running this program will print the constants in the left print zones and the variables in the right.

So far, we have merely been displaying the output right after we input data, one name and social security number for each run. But a very important computer operation would allow us to repeat the same instructions many times with one run.

GOTO

The GOTO statement creates a jump to a line number other than the next line number in sequence. If the jump is to a line above the GOTO statement, a loop has been created. The format of the GOTO statement is:

nn-1 GOTO nn-2

where: nn-1 is the line number of the statement

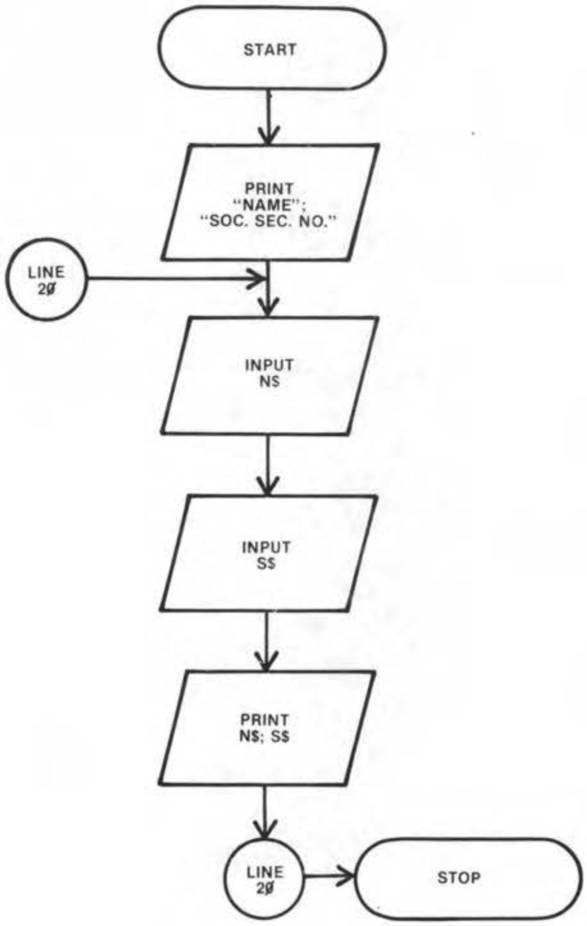
GOTO is the keyword and, nn-2 is the line number of the statement to be executed next.

Try this program:

```
1Ø PRINT "NAME"; "SOC.SEC.NO."
2Ø INPUT N$
3Ø INPUT S$
4Ø PRINT N$; S$
5Ø GOTO 2Ø
6Ø STOP
```

When this program is run, the constants on line 1Ø will be displayed first, lines 2Ø and 3Ø will wait for data to be entered, and the data entered will be printed by line 4Ø. But, unlike our previous programs, the program will not end there. Line 5Ø will cause a jump back to

line 2∅, where the INPUT statements will request that new values for the name field and the social security number field be entered. This process of inputting data and printing that data allows for a listing to be produced. Keep giving new names and numbers. As long as you don't fill up the screen, the program will continue to loop.



There is just one problem. Now that we have created a loop, there is no convenient way to get out of it; that is, we are "in a loop."

Line 6∅ will never be executed. Careful design of our program would have prevented this situation. Whenever you establish a loop in the program, you *must* provide an exit from it.

Looping is a necessary part of programming; being in a loop is not. Loops are useful, but must be controlled.

PROGRAM

```
1∅ PRINT "NAME"; "SOC.SEC.NO."
2∅ INPUT N$
3∅ INPUT S$
4∅ PRINT N$; S$
5∅ GOTO 2∅
6∅ STOP
```

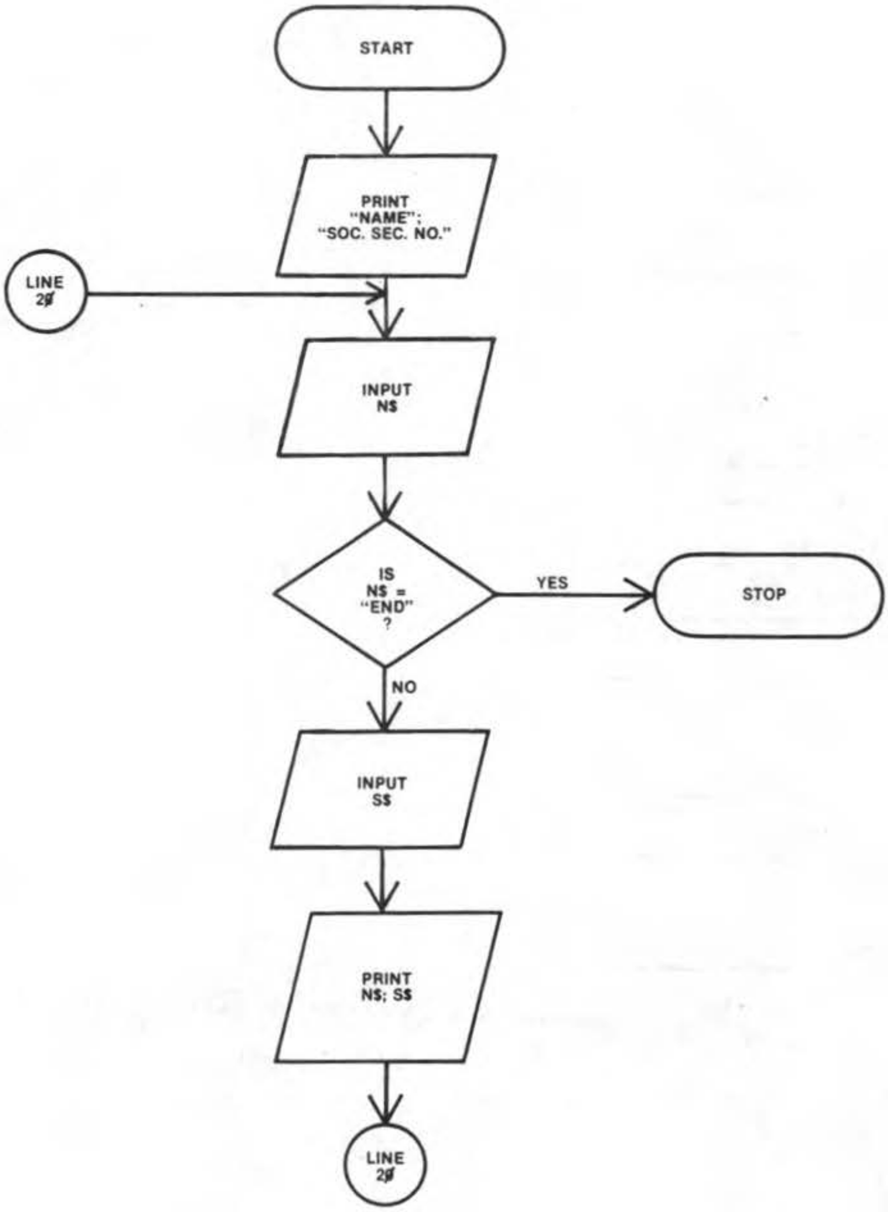
But, now that we are in a loop, let's see how we can get out of it.

1. Try pressing the BREAK key *immediately* after entering the social security number. If your timing is just right, you should see a report code indicating that the program was interrupted by the operator. If so, you can recall the program by pressing the ENTER key or by LISTing the program.

2. You should be able to fill the screen with names and social security numbers by repeatedly entering characters as prompted. Eventually the screen will fill up with data and a report code indicating this will appear on the bottom of the screen. Press ENTER or LIST to bring the program back.

3. If all else fails, the only remedy is to "pull the plug"; that is, power off and power back on. This, of course, results in the clearing of memory, so your program is lost. The only remedy is to reenter the program, an unpleasant experience for long programs, to say the least! "Caveat Coder"—Let the coder beware! As the saying goes, programs do not acquire bugs as people acquire germs—they are inserted there by the programmer.

A more proper design of our coding would have revealed the need for a decision box, one path of which would lead to line 6Ø or STOP.



Now you should have recalled or reentered the program we have been developing. Let's insert a line which will allow us to terminate the loop.

IF . . . THEN

The keyword IF poses a question to which the computer is able to respond with one of three possible answers: the two values are equal, the first value is less than the second value, or the first value is greater than the second. Depending on how the comparison is phrased, if the answer is *false*, the program will continue to execute the next statement in the normal sequence. If, however, the answer is true, something different will occur.

The word THEN is used to show what special event will occur. The format of the IF...THEN statement we need in order to control our loop is shown below:

nn IF string variable name = "constant" THEN GOTO nn

where: nn is a line number
IF is the keyword which compares a string variable to a constant (enclosed in quotes).

THEN begins the second half of the statement which shows that when, during execution, the values to the left are equal,

GOTO nn a branch to a line number will occur.

In our program, we will insert an IF...THEN statement which says that the control should branch to line 6Ø when the name entered (N$) is equal to the word END.

1. Put this line (25) into your program:

   25 IF N$ = "END" THEN GOTO 6Ø

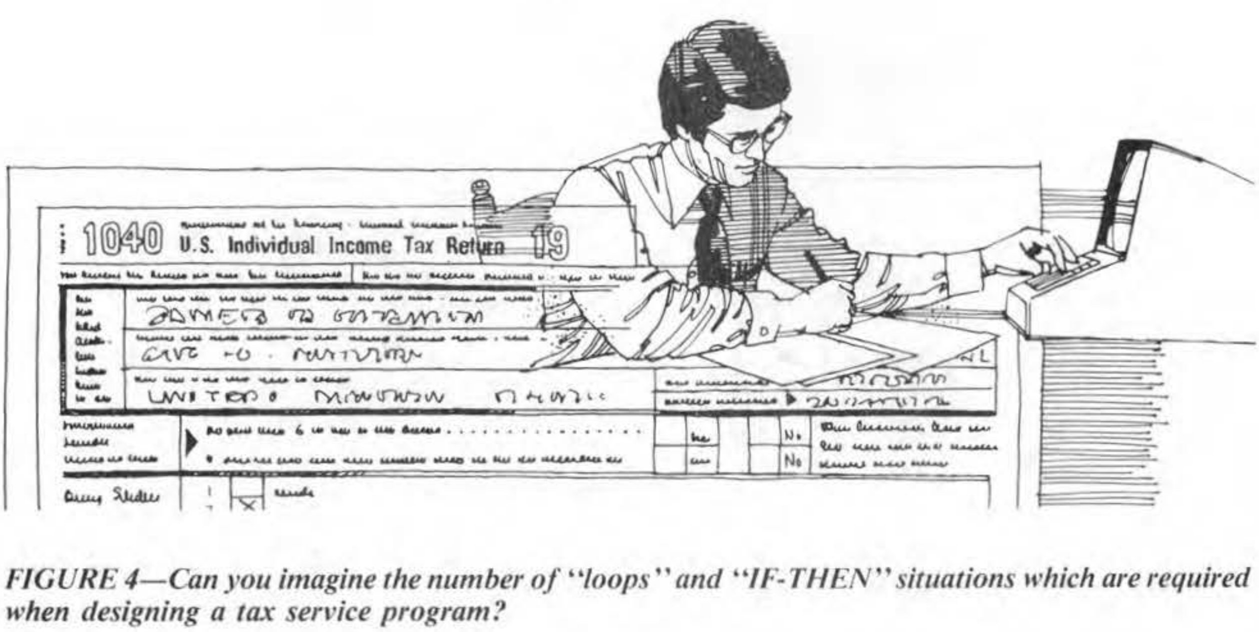
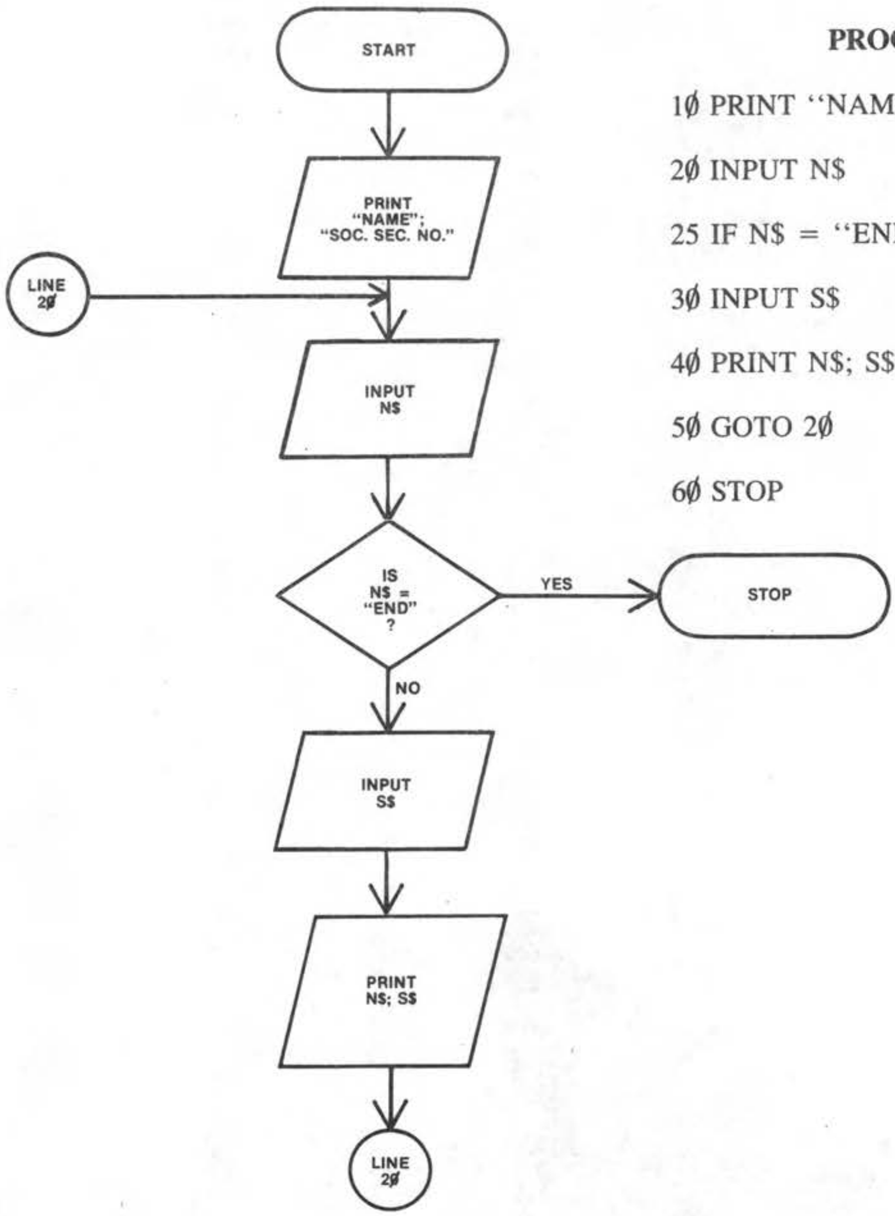


FIGURE 4—Can you imagine the number of "loops" and "IF-THEN" situations which are required when designing a tax service program?

This program now corresponds to the flowchart as it has been revised.



PROGRAM

```
1Ø PRINT "NAME"; "SOC.SEC.NO."
2Ø INPUT N$
25 IF N$ = "END" THEN GOTO 6Ø
3Ø INPUT S$
4Ø PRINT N$; S$
5Ø GOTO 2Ø
6Ø STOP
```

2. RUN the program again.
3. Enter two names and social security numbers.
4. When you are to enter a new name, enter END instead.

5. This time, a report code should appear showing that the program ended at line 6∅. We are not in a loop any more.

Remember, whenever creating a loop in a program, two things *must* be done:

1. A way out of the loop must be provided.

2. The way must actually occur. (If you never entered the name END, the program would continue to loop forever!)

REM Statement

Let's discuss one more BASIC statement before tackling your first programming assignment: the REM statement.

The REM statement is used to insert comments (or remarks) into our program. The computer will completely ignore REM statements; they are primarily useful so that humans can make some sense out of a program listing when:

1. It was written a long time ago or,

2. It was written by another programmer.

REM statements are generally written at the beginning of a program. They may give a name to the program and identify the author. They also can give a brief description of the variable names and logic used in the program.

The format for the REM statement is:

nn REM any characters

where: nn is the line number
REM is the keyword and any characters may be entered on the line.

In our previous program we might have inserted these REM statements so that our finished program reads:

```
1   REM NAME AND SOCIAL SECURITY
    NUMBER LISTING
2   REM (YOUR NAME)
3   REM N$ IS THE NAME FIELD
4   REM S$ IS THE SOCIAL SECURITY
    NUMBER FIELD
1∅  PRINT "NAME"; "SOC.SEC.NO."
2∅  INPUT N$
25  IF N$ = "END" THEN GOTO 6∅
3∅  INPUT S$
4∅  PRINT N$; S$
5∅  GOTO 2∅
6∅  STOP
```

NOTE: Main memory space is limited. If you should write a program that is too big to fit in RAM, first consider deleting the REM statements. Although remarks are ignored when the program is run, they do take up bytes of storage.

Now, see how much you have learned by completing the following Programmer's Check. Check your flowchart and program solutions and be sure that you understand the procedures before continuing.

PROGRAMMER'S CHECK

1

Creating A Phone List

Program Name: IU6A1 (Instruction Unit 6, Assignment 1)
Type: INPUT/OUTPUT
Specifications:

Produce a phone number listing on the screen by using INPUT statements. Three fields should be defined:

1. A name field

2. An area code field

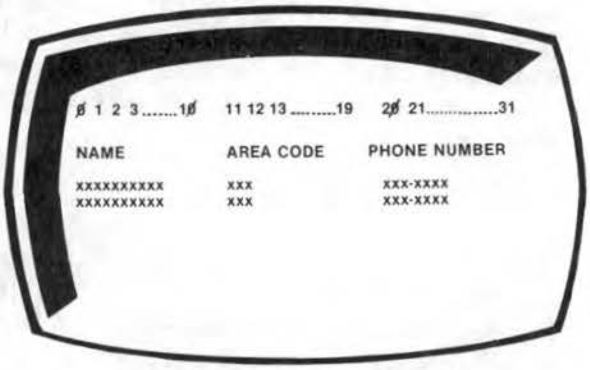
3. A phone number field

The programmer should choose his/her own variable names (make them all string variables; we don't need to do any calculations with these fields).

Constant, column headings should be printed at the top of the screen. Use the TAB function to begin each constant and variable in the proper format. Print zones (commas) won't work if you only have two zones per line!

Provide a way out of your program loop when the area code is 999.

OUTPUT FORMAT:



NOTE: NO NAME ENTRIES LARGER THAN 1ø CHARACTERS SHOULD BE ENTERED.

Follow these steps:

1. Analyze the specifications as shown on the CRT. Make sure you understand what it is you are expected to do.

2. Design your program by drawing a flowchart. Walk through the flowchart until you have a reasonable expectation that the program will work.

3. Code the program carefully, following the design of your flowchart.

4. RUN the program.

5. If it works, congratulations!

6. If it doesn't work, don't despair! Programs often have bugs in them. Patiently review your design and coding until you see your error. Look over the sample program again. This program is closely related to it.

7. Compare your flowchart and program to the ones on the indicated page. Yours may well be different. But, if it works, you can now consider yourself an applications programmer!

(Answers on Page 14)

ADDING CALCULATIONS

As important as input/output processing is to most computer applications, the addition of arithmetic capabilities allows us to have "instant" access to more data than we have submitted as input. Any arithmetic problem which can be reduced to basic algebra can be quickly and accurately calculated by our computer. In BASIC, we will explore five arithmetic operations:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation (raising to a power)

Whenever using variables in arithmetic statements, we must be sure that they are defined as numeric variables. (That is, variable names which do *not* end in a dollar sign.) The computer uses only the digits Ø through 9 in its calculations—letters and special characters have no place in arithmetic!

THE LET STATEMENT

The LET statement assigns a numeric value to a numeric variable. The value assignment depends on what is to the right of the equals sign (=). The format of the LET statement is:

nn LET numeric variable = arithmetic expression or literal

where: nn is the line number,
LET is the keyword,
numeric variable is the variable we wish to have a value assigned *to* and the arithmetic expression or literal contains the values to be used in the calculation.

SIMPLE ASSIGNMENT OF VALUES

A LET statement can be used to force a numeric variable to take on a specific value. For example, if we wished to use a fixed interest rate in a program such as .Ø8 (8% = 8/1ØØ), we could assign the variable "R" that value.

```
1Ø LET R = .Ø8
```

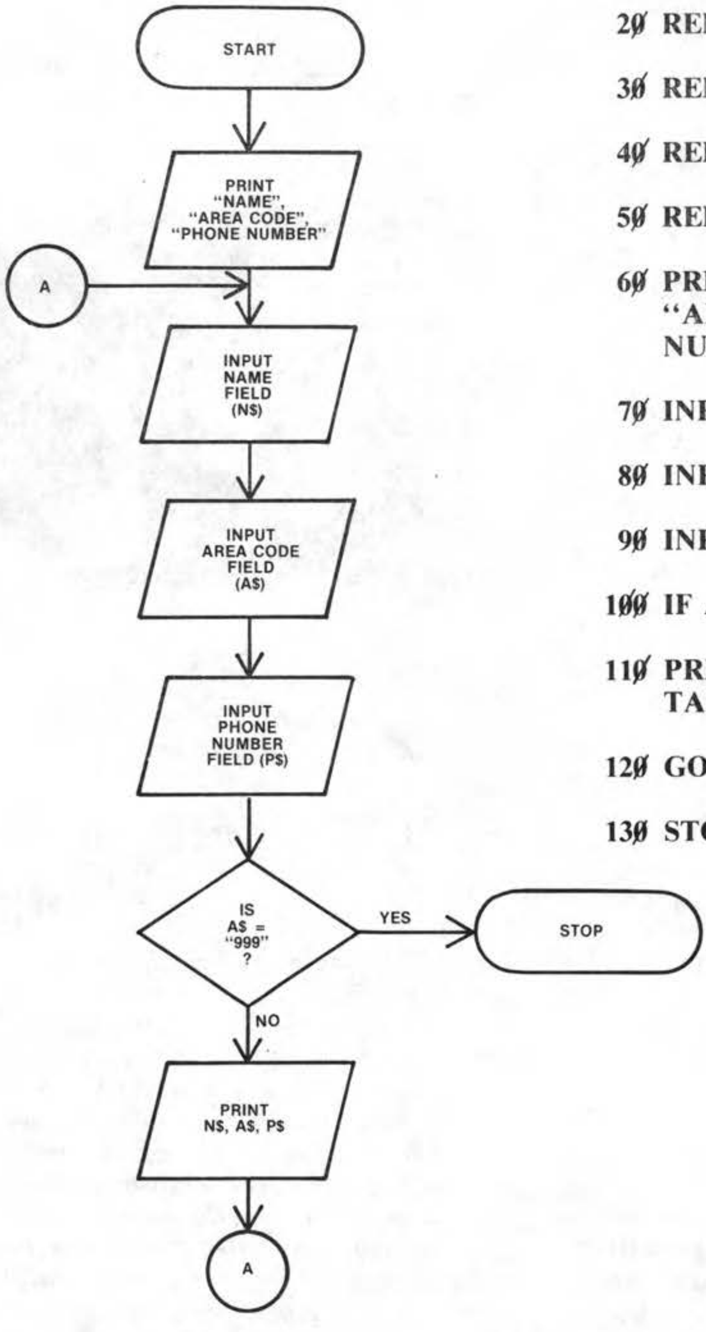
In this manner we could assign any value (number) to a numeric variable whenever we wish.

FIGURE 5—As in any other job, the programmer can suffer fatigue, strained eyes, numb fingers, back and neck problems, and poor circulation in the legs. To avoid these problems, work intelligently. Stop at least every hour and move around. Get away from your work area for a few minutes. Exercise and don't snack too much. Use proper lighting and position the CRT so there are no reflections. Change your position frequently. Always use good posture. Position the keyboard at the proper height for your arms and hands. And, when you are feeling too much strain or tiredness, rest.

PROGRAMMER'S CHECK ANSWERS

1

FLOWCHART SOLUTION TO 1U6A1



Program Solution to 1U6A1

```
1Ø REM PHONE NUMBER LISTING

2Ø REM YOUR NAME—1U6A1

3Ø REM N$....NAME

4Ø REM A$....AREA CODE

5Ø REM P$....PHONE NUMBER

6Ø PRINT TAB Ø; "NAME"; TAB 11;
   "AREA CODE"; TAB 18; "PHONE
   NUMBER"

7Ø INPUT N$

8Ø INPUT A$

9Ø INPUT P$

1ØØ IF A$ = "999" THEN GOTO 13Ø

11Ø PRINT TAB Ø; N$; TAB 13; A$;
    TAB 22; P$

12Ø GOTO 7Ø

13Ø STOP
```

NOTE: String variables can also be assigned values. They cannot, however, be used in arithmetic. Whenever a string variable is to be assigned a value, the value must be enclosed in quotes. The format is:

nn LET string variable = "character string"

where: nn is the line number
LET is the keyword
character string contains the characters that are being assigned to the variable.

An example would be:

```
1Ø LET N$ = "JOE"
```

ADDITION

Addition of two numeric values is accomplished by using the plus symbol (+) in an arithmetic expression. For example:

```
1Ø LET A = 2 + 3
```

would assign the value 5 to the numeric variable A. Statements like this should be avoided in a program, however. Having the computer do arithmetic on literals (actual values like 2 and 3) is a waste of computer time. We would be better off to do the calculation once ourselves rather than making the program recalculate over and over again. Replacing the above statement with this:

```
1Ø LET A = 5
```

would result in a much more efficient program.

Arithmetic expressions in a program almost always involve one or more numeric variables. Therefore, the result of the calculation will depend upon the value in the variable(s) when the program is running.

In an application involving maintenance of a savings account balance, we may be inputting several variables such as the beginning balance, deposits and withdrawals. One of our calculations will involve adding the deposits to the old balance. If we assign variable names of D to the deposits, O to the old balance and N to the new balance, our LET statement would be:

```
5Ø LET N = O + D
```

The values contained in RAM under the name "O" and "D" would be added together and the sum placed in RAM under the name "N."

SUBTRACTION

Subtraction is easily accomplished in BASIC through the use of the minus sign (—). The value to the right of the minus sign will be subtracted from the value to the left of the minus sign. Therefore:

```
1Ø LET A = 16 — 5
```

would result in the value 11 being placed into main storage as the value of "A."

Our savings account program requires that we subtract withdrawals from our old balance as part of the formula for calculating the new balance. We could insert a subtraction operation into our previous statement, to subtract W (withdrawals) from the sum of the old balance (O) and deposits (D):

```
5Ø LET N = O + D — W
```

DIVISION

Numeric values are divided by using the division symbol or slash (/). In this case:

```
1Ø LET A = 1Ø/5
```

"A" would assume the value of 2, or the result of ten divided by 5. If we were trying to determine an average game (A) score for a series of three bowling games (S), we would use the statement:

```
5 LET A = S/3
```

MULTIPLICATION

We can obtain the product of two values by utilizing the multiplication symbol or an asterisk (*). The statement:

```
5 LET A = 3 * 8
```

would give the variable "A" the value of 24. If we wanted to calculate our gross pay check amount (G) from the variables hours worked (H) and pay rate per hour (P) then:

```
5 LET G = H * P
```

would be valid.

EXPONENTIATION

We can raise a value to a power by using the double asterisk key (**). To raise the number 4 to the power of 2 (square it), the formula is:

```
5 LET A = 4 ** 2
```

which would result in A being assigned the value of 16 (4x4).

NOTE: We could find the square root of 4 by "raising" it to the power of one-half. Most versions of BASIC, however, have a square root function built into ROM.

EVALUATING A COMPLEX ARITHMETIC STATMENT

A complex arithmetic statement is one in which more than one operation is being performed. In a complex statement, it is important that we understand the order in which the computer will do the operations. Look at these LET statements:

```
3Ø LET B = 6
4Ø LET C = 2
5Ø LET D = 3
6Ø LET A = B + C * D
7Ø PRINT A
```

If we were to run this program, what would be the value of A? Look at one way in which the answer would be 24 (6 + 2 * 3 = 8 * 3 = 24). Another possibility would be 12 (6 + 2 * 3 = 6 + 6 = 12). Actually the correct answer is 12. The computer has a predetermined sequence for evaluating complex arithmetic statements. It follows:

1. First, any exponentiation is done.
2. Second, multiplications and divisions are done. If more than one of these is in a statement, the leftmost one is done first.
3. Lastly, additions and subtractions are executed, also from left to right.

This order of evaluation ensures that the results of a calculation produce unambiguous answers. In the example we just saw, the answer is 12 because the multiplication of variables C and D would take place before the addition of variable B. Another way of entering this statement would be to use parentheses, as in:

```
6Ø LET A = B + (C * D)
```

Parentheses override the normal sequence. That is, operations inside parentheses take place before any other operations are done. We generally would not use them, however, in the above statement when the normal order works. But if we wanted the result to be 24; that is, if we wanted to add B plus C before multiplying the sum times D, then we would have to use parentheses, as in:

```
6Ø LET A = (B + C) * D
```

See if you can figure out the answer to this statement:

```
1Ø LET B = 1Ø
2Ø LET C = 2
3Ø LET D = 1
4Ø LET E = 4
5Ø LET A = D + E ** C — B/C
6Ø PRINT A
7Ø STOP
```

When you have determined the answer, ENTER the above program and run it. If you have followed the proper sequence, both answers should agree: (12).

Now modify statement 5Ø, inserting parentheses only, so that it reads:

```
5Ø LET A = ((D + E) ** C — B)/C
```

The answer this time would be 7.5. The innermost parenthetical expression (D + E) is done first. Then this result (5) is raised to the power of C (5 ** 2 = 25) and B (1Ø) is subtracted from it. This is the calculation in the outer pair of parentheses. Finally, this result 15 is divided by C (2) or 15/2 = 7.5.

Be careful not to use parentheses indiscriminately as this leads to a confusing statement; only use them when you must.

ACCUMULATING AND FINAL TOTALS

Now we have seen how information not submitted as input can be obtained as output.

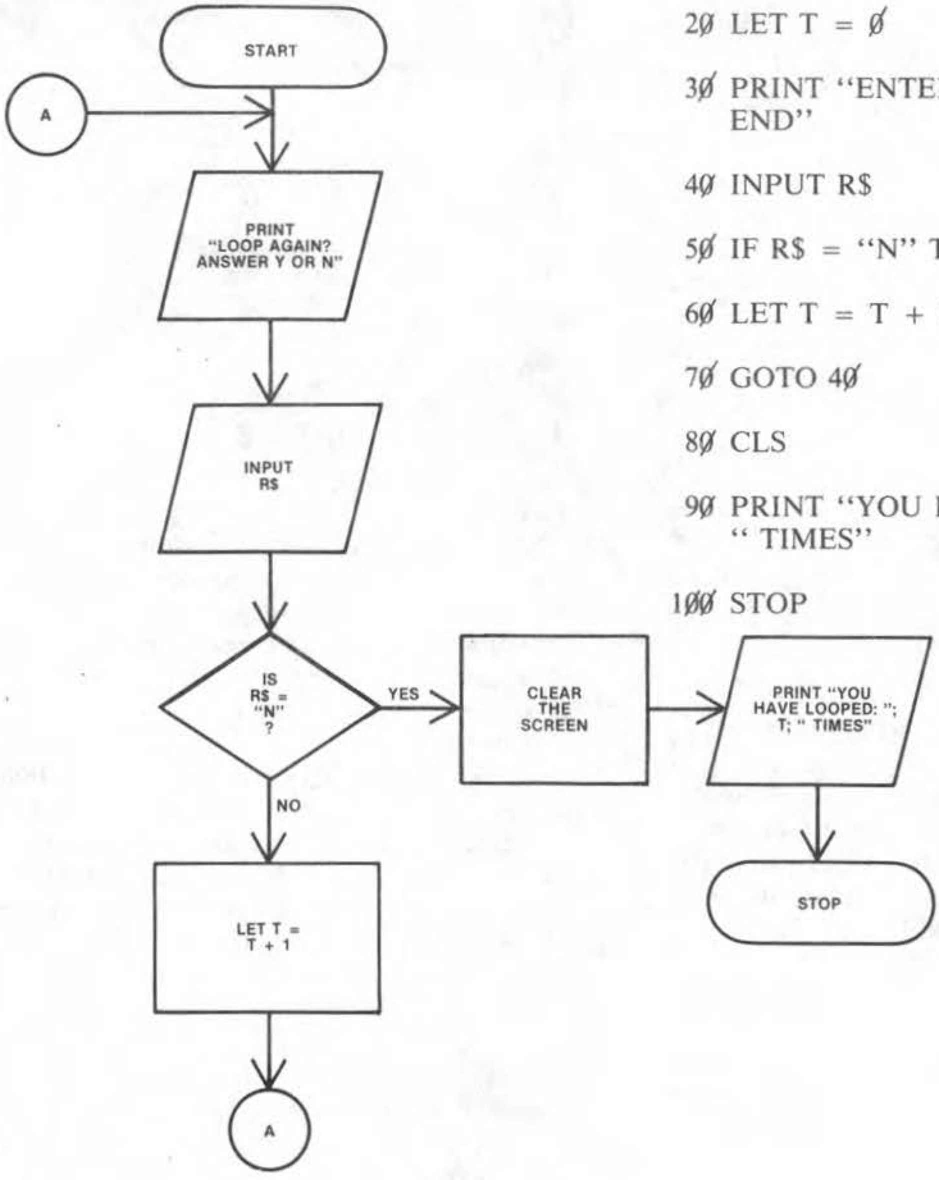
FIGURE 6—The programmer's job can take you many places. When a program just isn't working right, you sometimes have to go onto a job site in an attempt to identify all the program variables. A ten-minute talk with a construction foreman may save countless hours of frustration in trying to solve a job cost problem.

Sometimes, we desire to retrieve data that reflects information about all of the information that has been entered as input during the run of a program. Final totals relating to number of entries made, total of all deposits and withdrawals made this month, etc.

To serve this purpose, accumulators or counters are used by programmers. These are nothing more than numeric variables which are initially set at zero and added to every time an entry is made. After all entries have been made and the loop is completed, the final results can be displayed.

For an example, let's write a simple program which does nothing more than continually loop until we enter "NO." At the end of the program, the number of times we didn't respond with "NO" will be displayed.

The flowchart looks like this:



The program should be entered as follows:

```
10 REM ACCUMULATING TOTALS
20 LET T = 0
30 PRINT "ENTER Y TO LOOP, N TO
   END"
40 INPUT R$
50 IF R$ = "N" THEN GOTO 80
60 LET T = T + 1
70 GOTO 40
80 CLS
90 PRINT "YOU HAVE LOOPED: "; T;
   " TIMES"
100 STOP
```

RUN the program, and when prompted, enter "Y" (actually, anything other than "N" will work). When you tire of doing this, press N. The screen will be cleared and you will see a total of the number of times you looped.

The important statement is line 60: LET T = T + 1. Note that this is not an arithmetic *equation* we are trying to solve. Rather, it says, "Let T equal the value of T plus one." That is, every time T is through the loop, one will be added to the variable T. The results that are printed on line 90, however, will only be valid if we initialize the value of T to 0, as we did in line 20.

In most computers, moreover, our program won't work at all if we do not assign a value to T before using it in an arithmetic statement. So far, we have learned to give values to variables only through the use of the LET or INPUT statements. Some computers will automatically initialize numeric variables at zero and string variables as blank. To see how your computer works, delete line 2Ø from the previous program and RUN it again. More than likely, you will receive a report code indicating that a variable was used before a value was assigned to it for line 6Ø.

We can also accumulate more interesting final totals. Let's try another short program.

```
1Ø REM ADDING NUMBERS
2Ø LET T = Ø
3Ø INPUT A
4Ø IF A = Ø THEN GOTO 7Ø
5Ø LET T = T + A
6Ø GOTO 3Ø
7Ø PRINT "THE SUM IS: "; T
8Ø STOP
```

When you enter and run this program, you will repeatedly be prompted to enter values for the variable A. Any values other than Ø may be entered. Enter several numbers, of two or more digits each (a decimal point can be used if you wish to add up dollars and cents, as in 1ØØ.5Ø—don't use dollar signs or commas, however, as these are treated as special characters).

When you enter the value Ø, the program will branch at line 4Ø to line 7Ø where the sum will be displayed.

Hand calculators use logic like this to calculate and accumulate. They, however, cannot display their output in as pleasing a manner as we will learn to do.

Now, let's develop a program which should serve as an example of how we can incorporate arithmetic operations with input/output operations.

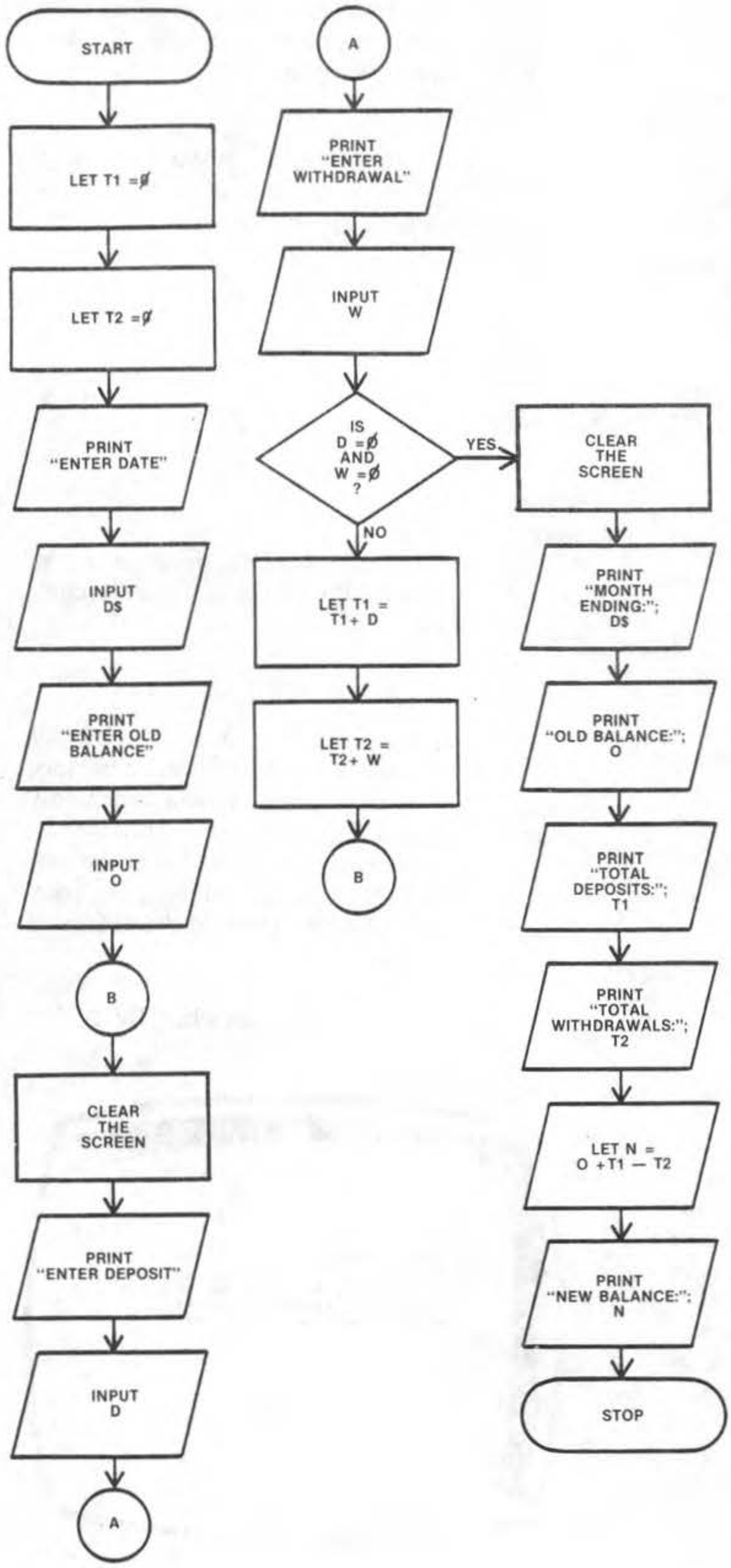
SAMPLE PROGRAM

In this example, we will write an application to do a monthly savings account balancing program.

Our program will prompt for three input fields: first it will ask for the beginning balance value and the date. Then, it will repeatedly prompt for any deposits and/or withdrawals. When zeros are entered for both values, our output will display the old balance, the total amount of our deposits and withdrawals, and our final balance.

The output format looks like this:

The flowchart design is:



The program:

```
1Ø REM SAVINGS ACCOUNT
   BALANCE
2Ø REM YOUR NAME—IU6S2
3Ø REM T1....TOTAL DEPOSITS
   ACCUMULATOR
4Ø REM T2....TOTAL WITHDRAWALS
   ACCUMULATOR
5Ø REM D$....MONTH END DATE
   MM/DD/YY
6Ø REM O....OLD BALANCE
7Ø REM D....DEPOSIT
8Ø REM W....WITHDRAWAL
9Ø REM N....NEW BALANCE
1ØØ LET T1 = Ø
11Ø LET T2 = Ø
12Ø PRINT "ENTER DATE"
13Ø INPUT D$
14Ø PRINT "ENTER OLD BALANCE"
15Ø INPUT O
16Ø CLS
17Ø PRINT "ENTER A DEPOSIT OR Ø"
18Ø INPUT D
19Ø PRINT "ENTER A WITHDRAWAL
    OR Ø"
2ØØ INPUT W
21Ø IF D = Ø AND W = Ø THEN
    GOTO 25Ø
22Ø LET T1 = T1 + D
23Ø LET T2 = T2 + W
24Ø GOTO 16Ø
25Ø CLS
26Ø PRINT "MONTH ENDING:"; D$
27Ø PRINT "OLD BALANCE:"; O
28Ø PRINT "TOTAL DEPOSITS:"; T1
29Ø PRINT "TOTAL
    WITHDRAWALS:"; T2
3ØØ LET N = O + T1—T2
31Ø PRINT "NEW BALANCE IS:"; N
32Ø STOP
```

Note some interesting features of this program. Read over the remark (REM) statements; you need not enter them yourself. We really start our program with the initialization of two accumulators: T1 and T2 to Ø.

We then prompt for two variables—the date (as MM/DD/YY) and the old balance. The program then repeatedly prompts for a deposit and/or a withdrawal. We may enter any values we wish, but when we enter a Ø for both variables, line 21Ø will cause a jump out of the loop to line 25Ø. There, the screen will be cleared and output displayed. Some of the output data will be exactly the same as the input (the date and old balance). Others are actually accumulators (total deposits and total withdrawals). And finally, we have one which is the result of one calculation (the new balance).

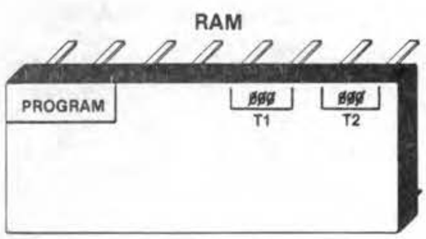
Try running the program after entering the following input:

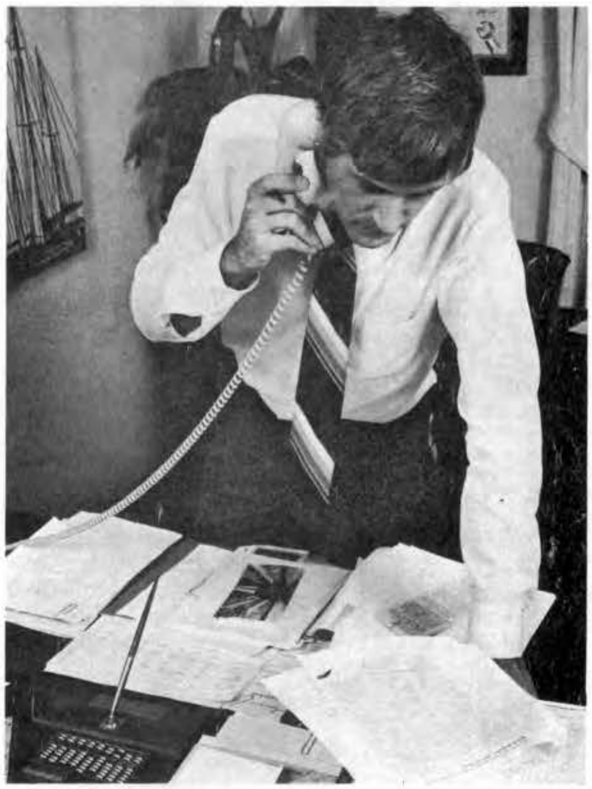
DATE	=	Ø1/31/83
OLD BALANCE	=	5ØØ.ØØ
DEPOSIT	=	5Ø.ØØ
WITHDRAWAL	=	25.5Ø
DEPOSIT	=	Ø
WITHDRAWAL	=	32.75
DEPOSIT	=	Ø
WITHDRAWAL	=	Ø

When all the data has been entered, the following should appear on the screen:

Let's walk through this program step-by-step to see how this output was achieved.

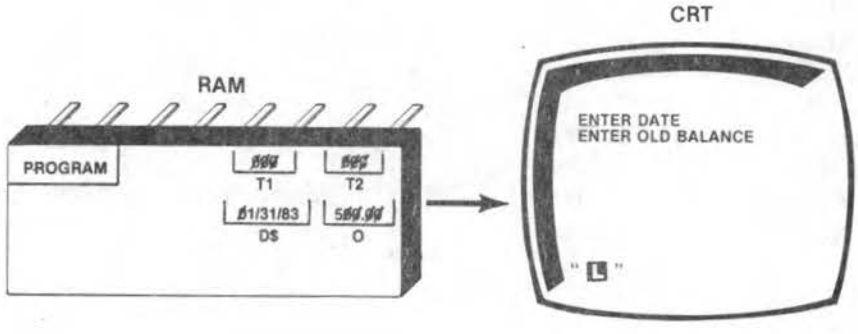
The remark statements are, if present, ignored by the computer. The first two LET statements (lines 1ØØ and 11Ø) cause two numeric variables to be established in RAM with values of Ø. With our program in storage as well, RAM looks like this:



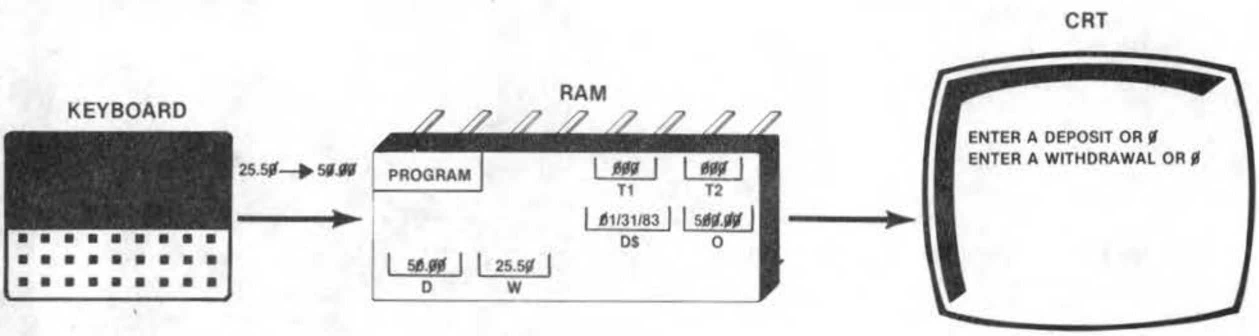


Computers do more work than people because they never have to answer the phone.

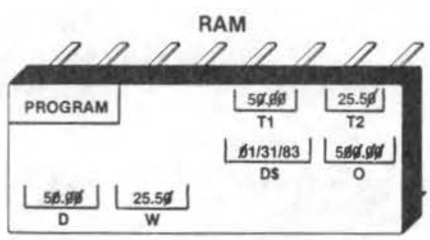
Lines 12Ø and 13Ø cause a line to be printed on the screen and a value for the date (D$) is prompted for. The next two statements (14Ø and 15Ø) do the same for the old balance and our system now looks like this:



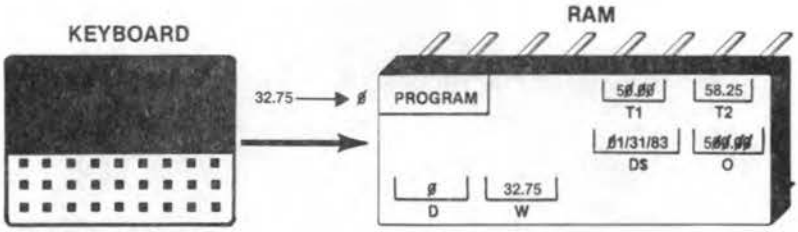
We now enter the loop. A deposit and a withdrawal are prompted for and give values to the variables D and W respectively.



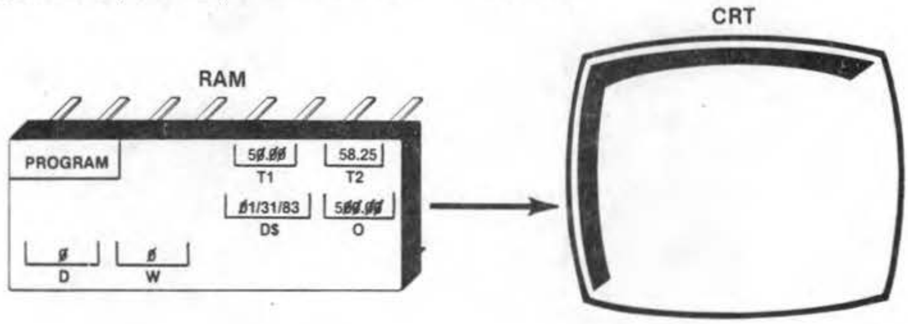
Since both D and W are not equal to Ø (line 21Ø will branch only if both are true), the next two lines will cause these two values to be accumulated in T1 and T2. The program then branches back to prompt for new values for D and W.



After a value of Ø is entered for deposit and 32.75 for withdrawal, both values will again be added to the accumulators.



When Ø values are entered for both a deposit and withdrawal, line 21Ø will cause a branch to line 25Ø where the screen will be cleared.



At this point, data from RAM will be displayed on the screen (lines 26Ø through 31Ø):

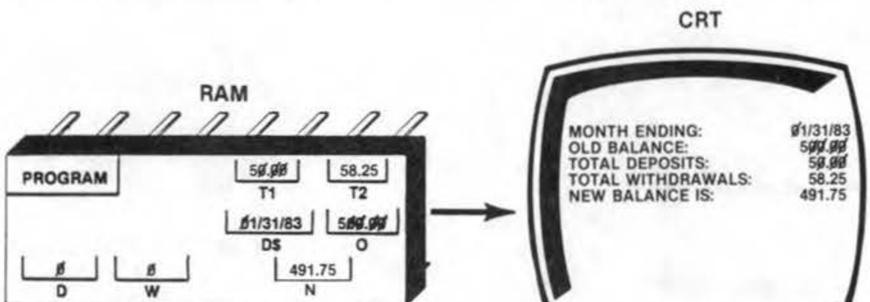
The constant MONTH ENDING: and the value in D$. (26Ø)

The constant OLD BALANCE: and the value entered for variable O. (27Ø)

The constant TOTAL DEPOSITS: and the variable T1 which has accumulated all D values entered. (28Ø)

The constant TOTAL WITHDRAWALS: and the variable T2 which is the sum of all the W values inputted. (29Ø)

The new balance is calculated (3ØØ) and then printed beside the constant NEW BALANCE IS: (31Ø)



Now try the following Programmer's Check on your own. Remember to work from a good design (flowchart!).

PROGRAMMER'S CHECK

2

Figuring Specifications

Program Name: IU6A2 (Instruction Unit 6, Assignment 2)

Type: ARITHMETIC

Specifications:

1. Produce a program which displays the area of a circular swimming pool as the radius changes. We are to produce a listing of areas for the ACE POOL CO. and client J. JONES. MR. JONES wishes to know how many square feet is in a pool of 5, 6, or 7 foot radii. The formula for calculating the area of a circle is Area = π * R ** 2 where π is the value 3.1415927, R is the radius which is to be squared (raised to the power of 2).

Your output should look like this:

Use input statements to get values for the company name, the client's name, and the radii.

(Answers on Pages 26 and 27)

2. Design and code a BASIC program which will calculate the cost of an item for sale at a discount, including a sales tax. For example, an item selling at \$25.00 with a 15% discount and a 6% sales tax would cost:

DISCOUNT $= \$25 \times \frac{15}{100} = \$3.75$

COST SUBJECT TO TAX =
\$25 — \$3.75 = \$21.25

SALES TAX $= \frac{6}{100}$ **X COST SUBJECT TO TAX (21.25) = \$1.28**

COST = \$21.25 + 1.28 = \$22.53

<u>OR:</u>

DISCOUNT = ORIGINAL COST TIMES DISCOUNT PERCENT

COST SUBJECT TO TAX = ORIGINAL COST MINUS DISCOUNT

SALES TAX = COST SUBJECT TO TAX TIMES TAX PERCENT

COST = COST SUBJECT TO TAX PLUS SALES TAX

(continued)

Programmer's Check 2 (continued)

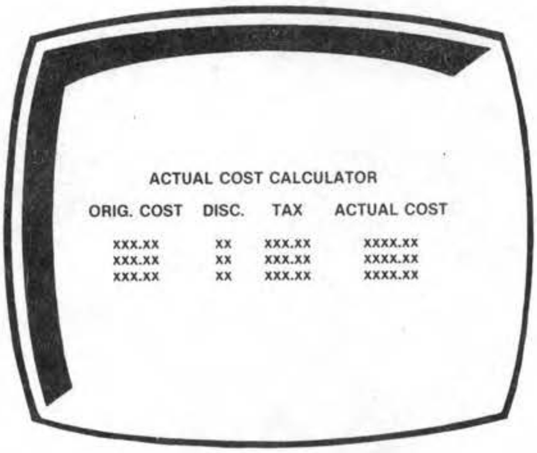
Each of these fields should be displayed on the screen for each item entered. If an original cost of Ø is entered, the program will display a new screen showing the number of items entered and the totals of the original cost, sales tax and final cost.

Program Name: **IU6A3 (Instruction Unit 6, Assignment 3)**

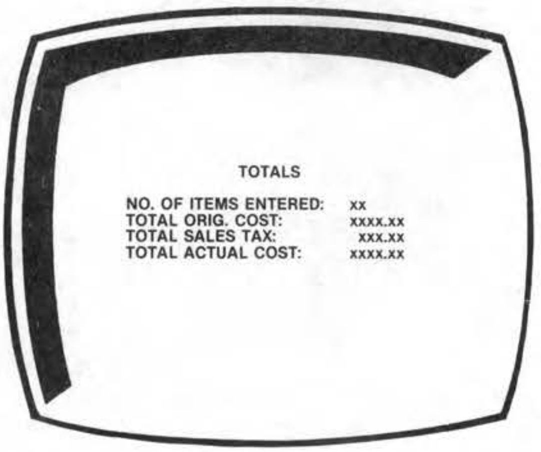
Type: **ARITHMETIC**

Specifications:

CRT OUTPUT



SECOND SCREEN



(Answers to #2 are found on Pages 28 and 29)

3. **Develop, design, and code a program which will generate output consisting of the number of cubic feet in a room. The program will consist of variables which will be input to a calculation of height times length times width.**

4. **Apply other formulas you know and use in your particular field of interest and develop software which will calculate and display the results in a pleasing format.**

Suggestions:

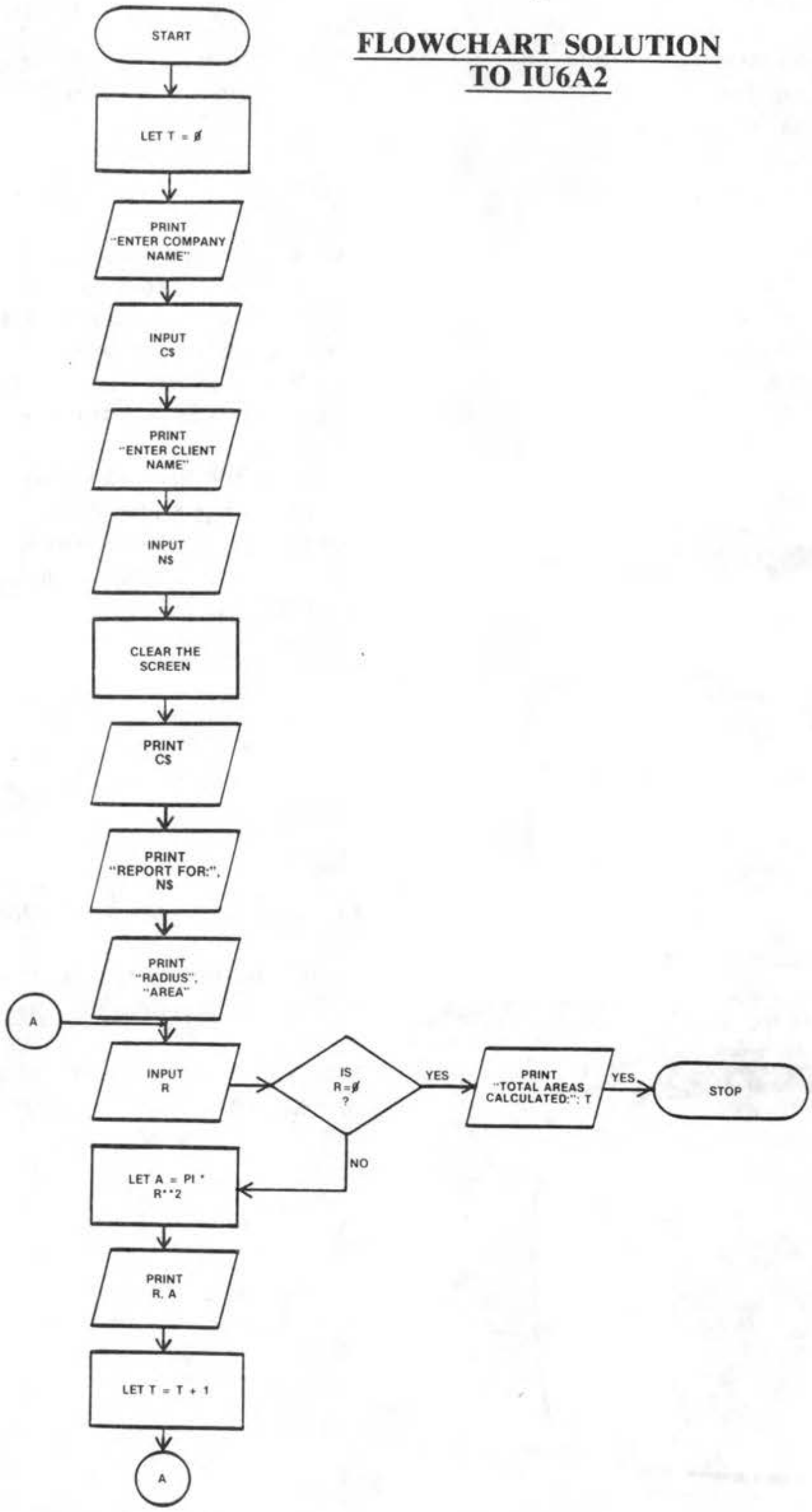
1. **Loan or mortgage amortization.**

2. **Areas, circumferences, and volumes of various geometrical objects.**

3. **Interest and return on various investments.**

2

FLOWCHART SOLUTION TO IU6A2



(continued)

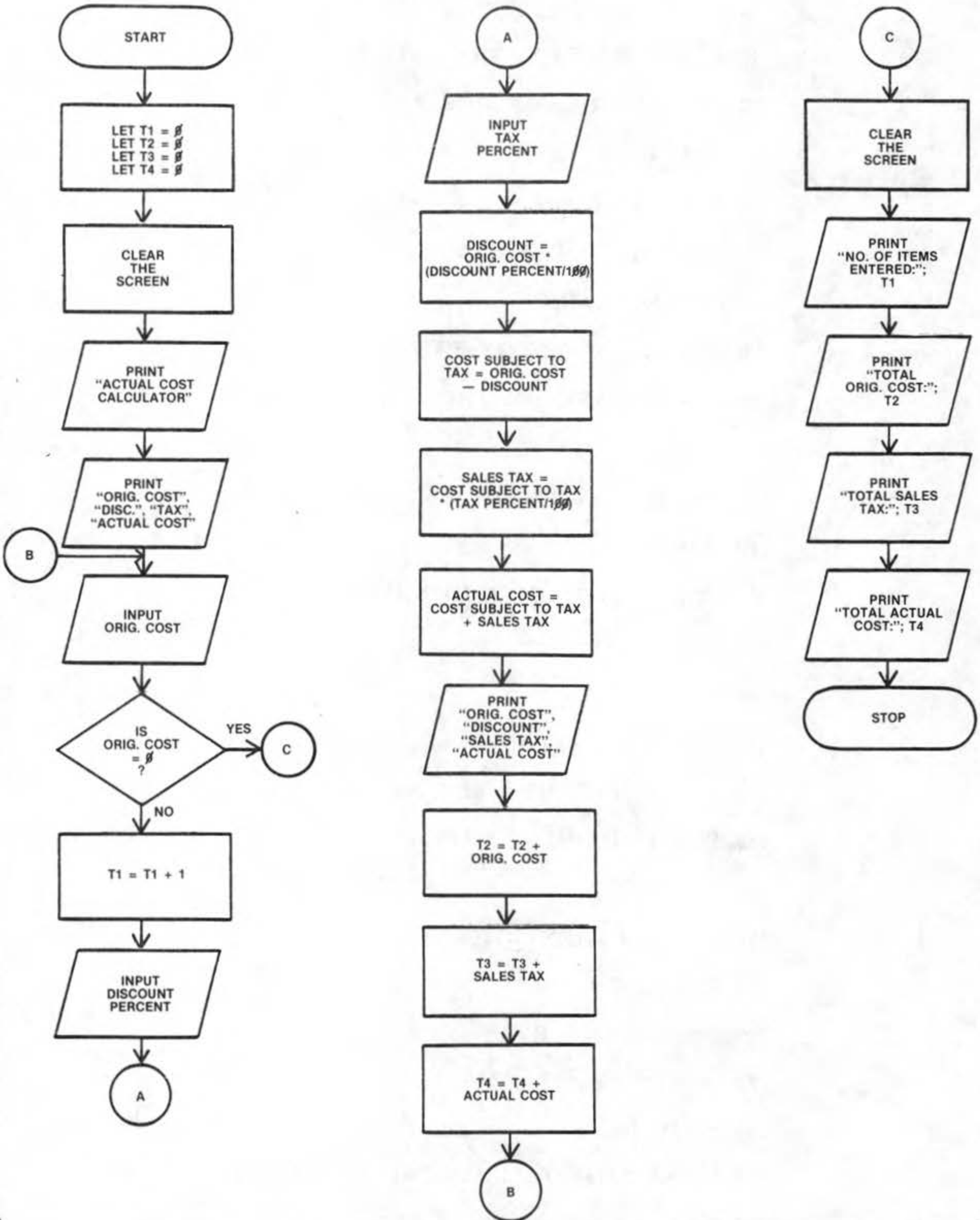
Programmer's Check 2 Answer (continued)

Program Solution to IU6A2

```
1Ø REM AREA OF VARIOUS POOL SIZES
2Ø REM YOUR NAME—IU6A2
3Ø REM T =....TOTAL NUMBER OF AREAS
4Ø REM C$....COMPANY NAME
5Ø REM N$...CLIENT NAME
6Ø REM R....RADIUS
7Ø REM PI... π OR CONSTANT 3.1415927
8Ø REM A....AREA OF THE POOL
85 LET T = Ø
1ØØ PRINT "ENTER COMPANY NAME"
11Ø INPUT C$
12Ø PRINT "ENTER CLIENT NAME"
13Ø INPUT N$
14Ø CLS
145 PRINT TAB 6; C$
15Ø PRINT "REPORT FOR:", N$
155 PRINT "RADIUS", "AREA"
16Ø INPUT R
17Ø IF R = Ø THEN GOTO 22Ø
18Ø LET A = PI * R ** 2
19Ø PRINT TAB 3; R, A
2ØØ LET T = T + 1
21Ø GOTO 16Ø
22Ø PRINT "TOTAL AREAS CALCULATED:"; T
23Ø STOP
```

Programmer's Check 2 Answer (continued)

FLOWCHART SOLUTION TO IU6A3



Programmer's Check 2 Answer (continued)

Program Solution to IU6A3

```
1Ø REM ACTUAL COST CALCULATOR
2Ø REM YOUR NAME—IU6A3
3Ø REM T1....TOTAL NO. OF ITEMS
4Ø REM T2....TOTAL ORIGINAL
   COST
5Ø REM T3....TOTAL SALES TAX
6Ø REM T4....TOTAL ACTUAL COST
7Ø REM C1....ORIGINAL COST
8Ø REM D1....DISCOUNT PERCENT
9Ø REM T....TAX PERCENT
1ØØ REM S....SALES TAX
11Ø REM C2....AMOUNT SUBJECT TO
   TAX
12Ø REM C3....ACTUAL COST
13Ø REM D2....DISCOUNT AMOUNT
14Ø LET T1 = Ø
15Ø LET T2 = Ø
16Ø LET T3 = Ø
17Ø LET T4 = Ø
18Ø CLS
19Ø PRINT TAB 4; "ACTUAL COST
   CALCULATOR"
2ØØ PRINT "ORIG. COST"; TAB 11;
   "DISC."; TAB 17; "TAX"; TAB 21;
   "ACTUAL COST"
21Ø INPUT C1
22Ø IF C1 = Ø THEN GOTO 35Ø
23Ø LET T1 = T1 + 1
24Ø INPUT D1
25Ø INPUT T
26Ø LET D2 = C1 * (D1/1ØØ)
27Ø LET C2 = C1—D2
28Ø LET S = C2 * (T/1ØØ)
29Ø LET C3 = C2 + S
3ØØ PRINT TAB 2; C1; TAB 11; D2;
   TAB 17; S; TAB 25; C3
31Ø LET T2 = T2 + C1
32Ø LET T3 = T3 + S
33Ø LET T4 = T4 + C3
34Ø GOTO 21Ø
35Ø CLS
36Ø PRINT "NO. OF ITEMS
   ENTERED:"; T1
37Ø PRINT "TOTAL ORIG. COST:"; T2
38Ø PRINT "TOTAL SALES TAX:"; T3
39Ø PRINT "TOTAL ACTUAL COST:";
   T4
4ØØ STOP
```

DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **How a program can be documented?**
  REM (remark) statements can be inserted into a BASIC program to help describe (to humans) the meanings of variables and the information regarding the name of the program and the author. Flowcharts can also be used to document the program.

- **How punctuation can control output?**
  In a PRINT statement, commas are used to display output in print zones, while semicolons are used to print in the next available space. Tab statements can be employed to more exactly define the positions of output variables and constants.

- **The difference between coding a loop and being in a loop?**
  Loops are necessary in programs as they allow for the repeated execution of statements. Loops are coded so that one run of a program can produce output for a number of different values entered as variables. Being in a loop means that no exit has been provided or taken and that the program will continue until some human intervenes.

- **How the computer evaluates an arithmetic formula?**
  The computer evaluates arithmetic statements from left to right, doing intermediate calculations in the following order:

  A. Expressions inside parentheses first
  B. Exponentiation
  C. Multiplication and division
  D. Addition and subtraction

SCHOOL OF COMPUTER TRAINING

EXAM 6

A Lesson in BASIC Programming

24706-2

Questions 1-20: *Circle the letter beside the one best answer to each question*

1. Which of the following is not a BASIC keyword?
   (a) GOTO (c) THEN
   (b) LET (d) PRINT

2. Which of the following is a valid name for a string variable?
   (a) 3$ (c) Q$
   (b) Cl (d) $

3. A string variable may take on which of the following values?
   (a) Letters of the alphabet
   (b) Numbers
   (c) Special characters
   (d) All of the above

4. A numeric variable may take on which of the following values?
   (a) Letters of the alphabet
   (b) Numbers
   (c) Special characters
   (d) All of the above

5. Loops in a program
   (a) are useful but must be controlled.
   (b) are useful but control themselves.
   (c) should always be avoided.
   (d) are a necessary evil.

6. Which of these methods cannot be used to control column spacing on the screen in a PRINT statement?
   (a) Tabs
   (b) Commas
   (c) Semicolons
   (d) Periods

7. An INPUT statement does which of the following when a program is run?
   (a) It causes a loop.
   (b) It prompts the user to enter data.
   (c) It accesses data already in the program.
   (d) It is ignored by the computer.

Questions 8-20: **The remaining questions pertain to a client who wants you to design and demonstrate a program about product specifications. Here is the problem:**

The Simpson Rug Weavers of Durham, N.C. produce Early American-style rugs in various diameters. The most popular diameters are 6 feet, 9 feet and 12 feet.

Mr. Simpson has hired you to create a computer program so that he can easily determine the square foot area of the various rug sizes. Since he needs the program quickly, he has asked that you supply a program immediately for the three most popular sizes.

Using the line number descriptions and flowchart illustration provided, answer the questions which follow. Take care when selecting answers. You may want to check and debug the completed program before transferring your answers onto the Answer Sheet.

HINT: The radius is determined by measuring a line from the exact center of the rug to the outer edge.

8. Line 7Ø would contain which of the following statements?

   (a) 7Ø REM D....DIAMETER
   (b) 7Ø REM C$
   (c) 7Ø REM PI....π OR CONSTANT 3.1415927
   (d) 7Ø REM D....OR CONSTANT 3.1415279

9. Item "E" on the flowchart should include one of the following commands:

   (a) INPUT N$
   (b) INPUT T
   (c) PRINT "RADIUS"; "AREA"
   (d) PRINT "CLEAR THE SCREEN"

10. Line 5Ø would include one of the following statements:

    (a) 5Ø REM T....TOTAL NUMBER OF AREAS
    (b) 5Ø REM A....AREA OF THE RUG
    (c) 5Ø REM N$....CLIENT NAME
    (d) 5Ø REM R....RADIUS

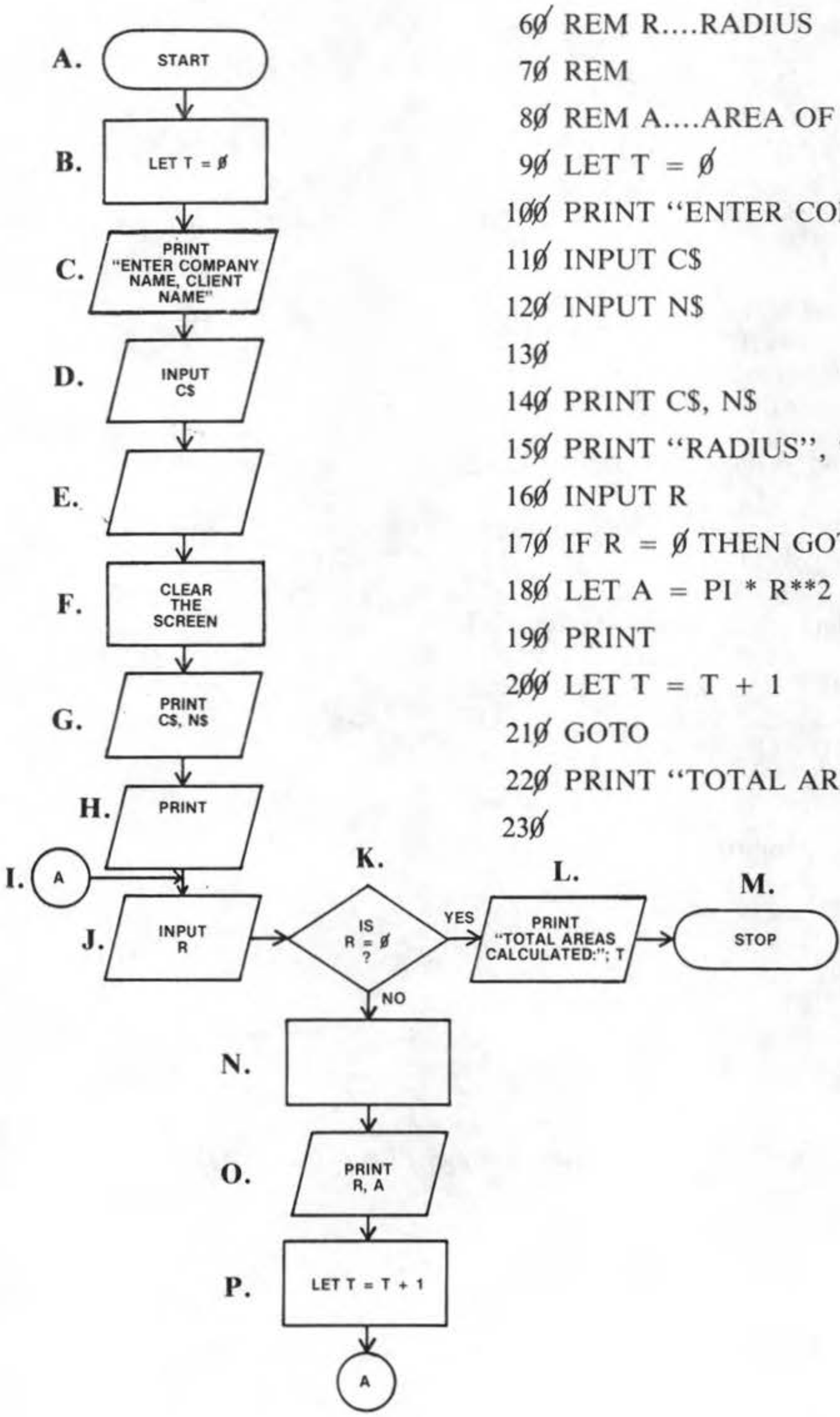
11. Only one of the commands given below is correct:

    (a) IF R = Ø, THEN GOTO 21Ø
    (b) IF R = Ø, THEN GOTO 19Ø
    (c) IF R = Ø, THEN GOTO 23Ø
    (d) IF R = Ø, THEN GOTO 22Ø

12. Item "H" is best completed by using one of the following commands:

    (a) PRINT "RADIUS", "AREA"
    (b) PRINT "TOTAL AREAS CALCULATED"
    (c) PRINT "CLEAR THE SCREEN"
    (d) PRINT "INPUT R"

```
1Ø REM AREA OF VARIOUS RUG SIZES
2Ø REM YOUR NAME
3Ø REM T....TOTAL NUMBER OF AREAS
4Ø REM C$....COMPANY NAME
5Ø REM
6Ø REM R....RADIUS
7Ø REM
8Ø REM A....AREA OF THE RUG
9Ø LET T = Ø
1ØØ PRINT "ENTER COMPANY NAME, CLIENT NAME"
11Ø INPUT C$
12Ø INPUT N$
13Ø
14Ø PRINT C$, N$
15Ø PRINT "RADIUS", "AREA"
16Ø INPUT R
17Ø IF R = Ø THEN GOTO
18Ø LET A = PI * R**2
19Ø PRINT
2ØØ LET T = T + 1
21Ø GOTO
22Ø PRINT "TOTAL AREAS CALCULATED:"; T
23Ø
```



13. Item "N" box on the flowchart can be made more complete by stating one of the following:

(a) LET A = PI * R**2
(b) LET A = INPUT R
(c) LET A = "TOTAL AREAS CALCULATED"
(d) LET A = T—T + T

14. Line 13Ø should be completed by including one of the following:

(a) INPUT N$
(b) INPUT R
(c) PRINT "ENTER COMPANY NAME"
(d) CLS

15. Line 23Ø is the same as

(a) item "P" on the flowchart.
(b) item "N" on the flowchart.
(c) item "M" on the flowchart.
(d) item "I" on the flowchart.

16. Line 21Ø can best be completed in one of the following ways:

(a) 21Ø GOTO 15Ø
(b) 21Ø GOTO 16Ø
(c) 21Ø GOTO 17Ø
(d) 21Ø GOTO 18Ø

17. The complete command for line 19Ø is one of the four below:

(a) 19Ø PRINT "RADIUS"; "AREA"
(b) 19Ø PRINT "TOTAL AREAS CALCULATED"
(c) 19Ø PRINT "STOP"
(d) 19Ø PRINT R, A

18. The formula used in this programming job is:

(a) AREA = PI * R**2
(b) DIAMETER = R **2 + PI
(c) AREA = PI * R * 2
(d) RADIUS = R * 2 * PI

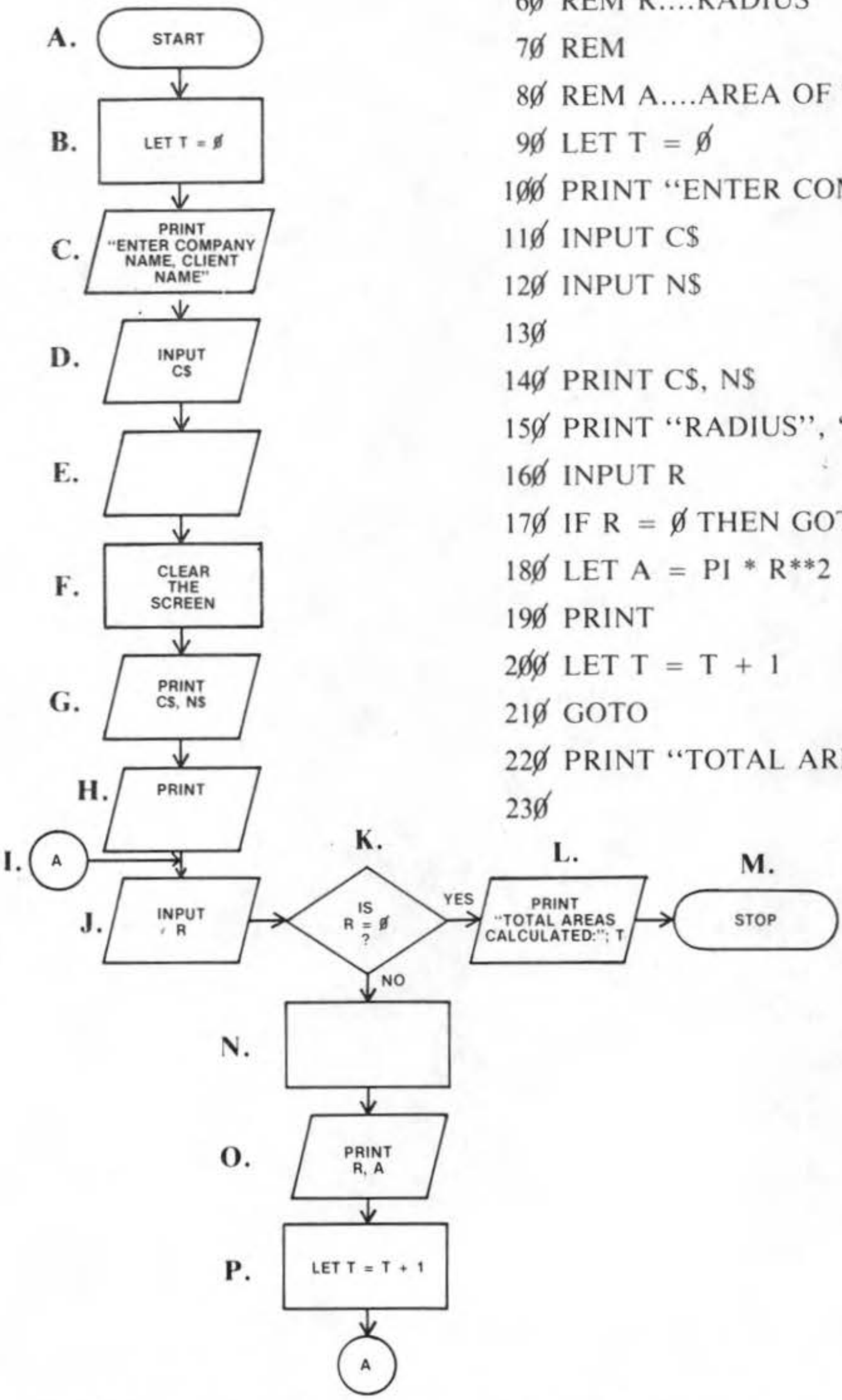
19. The "N$" is a symbol which stands for

(a) "No Money."
(b) company name.
(c) client's name.
(d) number of dollars.

20. Line 9Ø and Line 2ØØ

(a) should be identical for the program to work.
(b) should be changed to read: LET T = T + O.
(c) indicate the area of the rugs.
(d) are correct as stated.

```
1Ø REM AREA OF VARIOUS RUG SIZES
2Ø REM YOUR NAME
3Ø REM T....TOTAL NUMBER OF AREAS
4Ø REM C$....COMPANY NAME
5Ø REM
6Ø REM R....RADIUS
7Ø REM
8Ø REM A....AREA OF THE RUG
9Ø LET T = Ø
1ØØ PRINT "ENTER COMPANY NAME, CLIENT NAME"
11Ø INPUT C$
12Ø INPUT N$
13Ø
14Ø PRINT C$, N$
15Ø PRINT "RADIUS", "AREA"
16Ø INPUT R
17Ø IF R = Ø THEN GOTO
18Ø LET A = PI * R**2
19Ø PRINT
2ØØ LET T = T + 1
21Ø GOTO
22Ø PRINT "TOTAL AREAS CALCULATED:"; T
23Ø
```



WHEN YOU HAVE COMPLETED THE ENTIRE EXAM, TRANSFER YOUR ANSWERS TO THE ANSWER SHEET WHICH FOLLOWS.

ICS

ANSWER PAPER

To avoid delay, please insert all the details requested below

Subject PRACTICAL PROGRAMMING IN BASIC Course

Student's Reference (Letters / Figures)	Serial Number	Test No.	Ed No.	Tutor's Comments	Grade	Tutor
/	24706	6	2			

Name

Address

Post Code

After studying the foregoinc questions, record your answers in the matrix below by writing a cross (X), in ink, through the letter which you consider to be the correct answer. Sumbit ONLY this answer sheet to the School for correction DO NOT SUBMIT THE QUESTION SHEET. All questions must be attempted.

1.	A	B	C	D	E
2.	A	B	C	D	E
3.	A	B	C	D	E
4.	A	B	C	D	E
5.	A	B	C	D	E

6.	A	B	C	D	E
7.	A	B	C	D	E
8.	A	B	C	D	E
9.	A	B	C	D	E
10.	A	B	C	D	E

11.	A	B	C	D	E
12.	A	B	C	D	E
13.	A	B	C	D	E
14.	A	B	C	D	E
15.	A	B	C	D	E

16.	A	B	C	D	E
17.	A	B	C	D	E
18.	A	B	C	D	E
19.	A	B	C	D	E
20.	A	B	C	D	E

CUT HERE